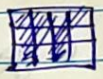
DSA - JS

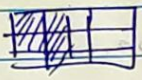
4) Big O : intro

→ big O is a way to mathematically figure out which of these two is better, which one run more efficiently

→ The code to run as quickly as possible be a efficient as possible
So by this measure, code one is the better code (take less time)
This is called time complexity.
→ we measure it in the no. of operations

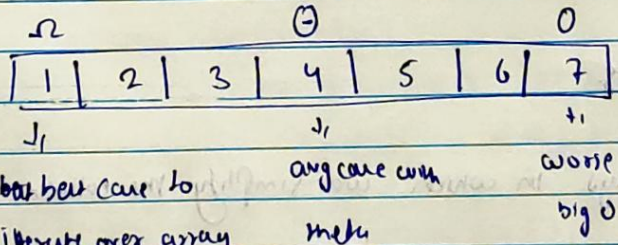
⇒ Space complexity : Space complexity is the amount of memory that something uses.

Code 1 → 15 min →  → more memory

Code 2 → 30 min →  → less memory

5) Big O : worst case

Ω → omega Θ → theta O → omicron (big O)



→ If we measuring big O that means always a worse case.

6) Big O : O(n) : first big O notation

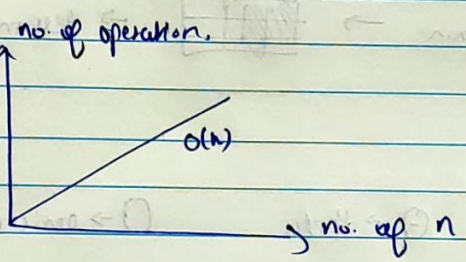
```
function logItems(n) {
    for(let i=0 ; i<n ; i++) {
        console.log(i)
    }
}
```

O(n)

logItem(10) → O(10) → it will run 10 times.

→ we pass the function the no. n and this ran n times.
→ O(n) is always a straight line.

→ The no. of operation is going to be proportional to whatever n is



7) Big O : Drop constants.

→ Big O has several ways in which we simplify the notation.

Drop constants

```
function logItems(n){
    ~~Console.log(i~~
    for (let i=0; i<n ; i++){
        Console.log(i)          O(n)
    }

    for (let j=0; j<n ; j++){
        Console.log(j)
    }                           O(n)
}
logItem(3)
```

$O(n) + O(n) \Rightarrow O(2n)$

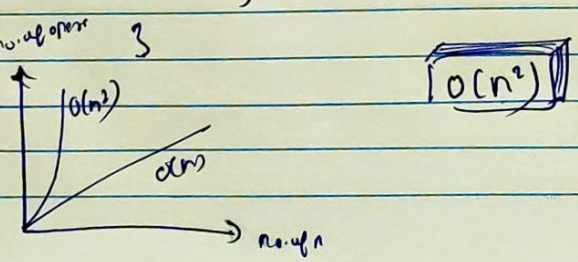
drop constants

→ no matter $O(n)$, $O(2n)$ .. $O(3n)$ .. $O(100n) \approx O(n)$
we drop the constant and we say code is $O(n)$ only

*) so our first rule for simplifying our big O notation drop constant.

8) Big O : $O(n*2)$

```
function logItems(n)
    for(let i=0; i<n; i++){          n
        for (let j=0; j<n; j++) {    *
                                     n
                                     ||
            Console.log(i,j)         n2
    }
}
```



→ $O(n^2)$